

39.1 AAA—ASCII Adjust After Addition

Opcode	Instruction	Description
37	AAA	ASCII adjust AL after addition

Description

Adjusts the sum of two unpacked BCD values to create an unpacked BCD result. The AL register is the implied source and destination operand for this instruction. The AAA instruction is only useful when it follows an ADD instruction that adds (binary addition) two unpacked BCD values and stores a byte result in the AL register. The AAA instruction then adjusts the contents of the AL register to contain the correct 1-digit unpacked BCD result.

If the addition produces a decimal carry, the AH register is incremented by 1, and the CF and AF flags are set. If there was no decimal carry, the CF and AF flags are cleared and the AH register is unchanged. In either case, bits 4 through 7 of the AL register are cleared to 0.

Operation

```
IF ((AL AND 0FH) > 9) OR (AF = 1)
  THEN
    AL ← (AL + 6);
    AH ← AH + 1;
    AF ← 1;
    CF ← 1;
  ELSE
    AF ← 0;
    CF ← 0;
FI;
AL ← AL AND 0FH;
```

Flags Affected

The AF and CF flags are set to 1 if the adjustment results in a decimal carry; otherwise they are cleared to 0. The OF, SF, ZF, and PF flags are undefined.

Exceptions (All Operating Modes)

None.

39.2 AAD—ASCII Adjust AX Before Division

Opcode	Instruction	Description
D5 0A	AAD	ASCII adjust AX before division
D5 <i>ib</i>	(No mnemonic)	Adjust AX before division to number base <i>imm8</i>

Description

Adjusts two unpacked BCD digits (the least-significant digit in the AL register and the most-significant digit in the AH register) so that a division operation performed on the result will yield a correct unpacked BCD value. The AAD instruction is only useful when it precedes a DIV instruction that divides (binary division) the adjusted value in the AX register by an unpacked BCD value.

The AAD instruction sets the value in the AL register to $(AL + (10 * AH))$, and then clears the AH register to 00H. The value in the AX register is then equal to the binary equivalent of the original unpacked two-digit (base 10) number in registers AH and AL.

The generalized version of this instruction allows adjustment of two unpacked digits of any number base (see the “Operation” section below), by setting the *imm8* byte to the selected number base (for example, 08H for octal, 0AH for decimal, or 0CH for base 12 numbers). The AAD mnemonic is interpreted by all assemblers to mean adjust ASCII (base 10) values. To adjust values in another number base, the instruction must be hand coded in machine code (D5 *imm8*).

Operation

```
tempAL ← AL;
tempAH ← AH;
AL ← (tempAL + (tempAH * imm8)) AND FFH; (* imm8 is set to 0AH for the AAD mnemonic *)
AH ← 0
```

The immediate value (*imm8*) is taken from the second byte of the instruction.

Flags Affected

The SF, ZF, and PF flags are set according to the result; the OF, AF, and CF flags are undefined.

Exceptions (All Operating Modes)

None.

39.3 AAM—ASCII Adjust AX After Multiply

Opcode	Instruction	Description
D4 0A	AAM	ASCII adjust AX after multiply
D4 <i>ib</i>	(No mnemonic)	Adjust AX after multiply to number base <i>imm8</i>

Description

Adjusts the result of the multiplication of two unpacked BCD values to create a pair of unpacked (base 10) BCD values. The AX register is the implied source and destination operand for this instruction. The AAM instruction is only useful when it follows a MUL instruction that multiplies (binary multiplication) two unpacked BCD values and stores a word result in the AX register. The AAM instruction then adjusts the contents of the AX register to contain the correct 2-digit unpacked (base 10) BCD result.

The generalized version of this instruction allows adjustment of the contents of the AX to create two unpacked digits of any number base (see the “Operation” section below). Here, the *imm8* byte is set to the selected number base (for example, 08H for octal, 0AH for decimal, or 0CH for base 12

numbers). The AAM mnemonic is interpreted by all assemblers to mean adjust to ASCII (base 10) values. To adjust to values in another number base, the instruction must be hand coded in machine code (D4 *imm8*).

Operation

```
tempAL ← AL;
AH ← tempAL / imm8; (* imm8 is set to 0AH for the AAD mnemonic *)
AL ← tempAL MOD imm8;
```

The immediate value (*imm8*) is taken from the second byte of the instruction.

Flags Affected

The SF, ZF, and PF flags are set according to the result. The OF, AF, and CF flags are undefined.

Exceptions (All Operating Modes)

None with the default immediate value of 0AH. If, however, an immediate value of 0 is used, it will cause a #DE (divide error) exception.

39.4 AAS—ASCII Adjust AL After Subtraction

Opcode	Instruction	Description
3F	AAS	ASCII adjust AL after subtraction

Description

Adjusts the result of the subtraction of two unpacked BCD values to create a unpacked BCD result. The AL register is the implied source and destination operand for this instruction. The AAS instruction is only useful when it follows a SUB instruction that subtracts (binary subtraction) one unpacked BCD value from another and stores a byte result in the AL register. The AAA instruction then adjusts the contents of the AL register to contain the correct 1-digit unpacked BCD result.

If the subtraction produced a decimal carry, the AH register is decremented by 1, and the CF and AF flags are set. If no decimal carry occurred, the CF and AF flags are cleared, and the AH register is unchanged. In either case, the AL register is left with its top nibble set to 0.

Operation

```
IF ((AL AND 0FH) > 9) OR (AF = 1)
THEN
    AL ← AL - 6;
    AH ← AH - 1;
    AF ← 1;
    CF ← 1;
ELSE
    CF ← 0;
    AF ← 0;
FI;
AL ← AL AND 0FH;
```

Flags Affected

The AF and CF flags are set to 1 if there is a decimal borrow; otherwise, they are cleared to 0. The OF, SF, ZF, and PF flags are undefined.

Exceptions (All Operating Modes)

None.

39.5 ADC—Add with Carry

Opcode	Instruction	Description
14 <i>ib</i>	ADC AL, <i>imm8</i>	Add with carry <i>imm8</i> to AL
15 <i>iw</i>	ADC AX, <i>imm16</i>	Add with carry <i>imm16</i> to AX
15 <i>id</i>	ADC EAX, <i>imm32</i>	Add with carry <i>imm32</i> to EAX
80 /2 <i>ib</i>	ADC <i>r/m8</i> , <i>imm8</i>	Add with carry <i>imm8</i> to <i>r/m8</i>
81 /2 <i>iw</i>	ADC <i>r/m16</i> , <i>imm16</i>	Add with carry <i>imm16</i> to <i>r/m16</i>
81 /2 <i>id</i>	ADC <i>r/m32</i> , <i>imm32</i>	Add with CF <i>imm32</i> to <i>r/m32</i>
83 /2 <i>ib</i>	ADC <i>r/m16</i> , <i>imm8</i>	Add with CF sign-extended <i>imm8</i> to <i>r/m16</i>
83 /2 <i>ib</i>	ADC <i>r/m32</i> , <i>imm8</i>	Add with CF sign-extended <i>imm8</i> into <i>r/m32</i>
10 /r	ADC <i>r/m8</i> , <i>r8</i>	Add with carry byte register to <i>r/m8</i>
11 /r	ADC <i>r/m16</i> , <i>r16</i>	Add with carry <i>r16</i> to <i>r/m16</i>
11 /r	ADC <i>r/m32</i> , <i>r32</i>	Add with CF <i>r32</i> to <i>r/m32</i>
12 /r	ADC <i>r8</i> , <i>r/m8</i>	Add with carry <i>r/m8</i> to byte register
13 /r	ADC <i>r16</i> , <i>r/m16</i>	Add with carry <i>r/m16</i> to <i>r16</i>
13 /r	ADC <i>r32</i> , <i>r/m32</i>	Add with CF <i>r/m32</i> to <i>r32</i>

Description

Adds the destination operand (first operand), the source operand (second operand), and the carry (CF) flag and stores the result in the destination operand. The destination operand can be a register or a memory location; the source operand can be an immediate, a register, or a memory location. (However, two memory operands cannot be used in one instruction.) The state of the CF flag represents a carry from a previous addition. When an immediate value is used as an operand, it is sign-extended to the length of the destination operand format.

The ADC instruction does not distinguish between signed or unsigned operands. Instead, the processor evaluates the result for both data types and sets the OF and CF flags to indicate a carry in the signed or unsigned result, respectively. The SF flag indicates the sign of the signed result.

The ADC instruction is usually executed as part of a multibyte or multiword addition in which an ADD instruction is followed by an ADC instruction.

Operation

$DEST \leftarrow DEST + SRC + CF;$

Flags Affected

The OF, SF, ZF, AF, CF, and PF flags are set according to the result.

Protected Mode Exceptions

#GP(0) If the destination is located in a nonwritable segment.

If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

If the DS, ES, FS, or GS register is used to access memory and it contains a null segment selector.

- #SS(0) If a memory operand effective address is outside the SS segment limit.
- #PF(fault-code) If a page fault occurs.
- #AC(0) If alignment checking is enabled and an unaligned memory reference is made while the current privilege level is 3.

Real-Address Mode Exceptions

- #GP If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.
- #SS If a memory operand effective address is outside the SS segment limit.

Virtual-8086 Mode Exceptions

- #GP(0) If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.
- #SS(0) If a memory operand effective address is outside the SS segment limit.
- #PF(fault-code) If a page fault occurs.
- #AC(0) If alignment checking is enabled and an unaligned memory reference is made.

39.6 ADD—Add

Opcode	Instruction	Description
04 <i>ib</i>	ADD AL, <i>imm8</i>	Add <i>imm8</i> to AL
05 <i>iw</i>	ADD AX, <i>imm16</i>	Add <i>imm16</i> to AX
05 <i>id</i>	ADD EAX, <i>imm32</i>	Add <i>imm32</i> to EAX
80 /0 <i>ib</i>	ADD <i>r/m8</i> , <i>imm8</i>	Add <i>imm8</i> to <i>r/m8</i>
81 /0 <i>iw</i>	ADD <i>r/m16</i> , <i>imm16</i>	Add <i>imm16</i> to <i>r/m16</i>
81 /0 <i>id</i>	ADD <i>r/m32</i> , <i>imm32</i>	Add <i>imm32</i> to <i>r/m32</i>
83 /0 <i>ib</i>	ADD <i>r/m16</i> , <i>imm8</i>	Add sign-extended <i>imm8</i> to <i>r/m16</i>
83 /0 <i>ib</i>	ADD <i>r/m32</i> , <i>imm8</i>	Add sign-extended <i>imm8</i> to <i>r/m32</i>
00 /r	ADD <i>r/m8</i> , <i>r8</i>	Add <i>r8</i> to <i>r/m8</i>
01 /r	ADD <i>r/m16</i> , <i>r16</i>	Add <i>r16</i> to <i>r/m16</i>
01 /r	ADD <i>r/m32</i> , <i>r32</i>	Add <i>r32</i> to <i>r/m32</i>
02 /r	ADD <i>r8</i> , <i>r/m8</i>	Add <i>r/m8</i> to <i>r8</i>
03 /r	ADD <i>r16</i> , <i>r/m16</i>	Add <i>r/m16</i> to <i>r16</i>
03 /r	ADD <i>r32</i> , <i>r/m32</i>	Add <i>r/m32</i> to <i>r32</i>

Description

Adds the first operand (destination operand) and the second operand (source operand) and stores the result in the destination operand. The destination operand can be a register or a memory location; the source operand can be an immediate, a register, or a memory location. (However, two memory operands cannot be used in one instruction.) When an immediate value is used as an operand, it is sign-extended to the length of the destination operand format.

The ADD instruction does not distinguish between signed or unsigned operands. Instead, the processor evaluates the result for both data types and sets the OF and CF flags to indicate a carry in the signed or unsigned result, respectively. The SF flag indicates the sign of the signed result.

Operation

$DEST \leftarrow DEST + SRC;$

Flags Affected

The OF, SF, ZF, AF, CF, and PF flags are set according to the result.

Protected Mode Exceptions

- #GP(0) If the destination is located in a nonwritable segment.
If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.
If the DS, ES, FS, or GS register is used to access memory and it contains a null segment selector.
- #SS(0) If a memory operand effective address is outside the SS segment limit.
- #PF(fault-code) If a page fault occurs.
- #AC(0) If alignment checking is enabled and an unaligned memory reference is made while the current privilege level is 3.

Real-Address Mode Exceptions

- #GP If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.
- #SS If a memory operand effective address is outside the SS segment limit.

Virtual-8086 Mode Exceptions

- #GP(0) If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.
- #SS(0) If a memory operand effective address is outside the SS segment limit.
- #PF(fault-code) If a page fault occurs.
- #AC(0) If alignment checking is enabled and an unaligned memory reference is made.

39.7 AND—Logical AND

Opcode	Instruction	Description
24 <i>ib</i>	AND AL, <i>imm8</i>	AL AND <i>imm8</i>
25 <i>iw</i>	AND AX, <i>imm16</i>	AX AND <i>imm16</i>
25 <i>id</i>	AND EAX, <i>imm32</i>	EAX AND <i>imm32</i>
80 /4 <i>ib</i>	AND <i>r/m8</i> , <i>imm8</i>	<i>r/m8</i> AND <i>imm8</i>
81 /4 <i>iw</i>	AND <i>r/m16</i> , <i>imm16</i>	<i>r/m16</i> AND <i>imm16</i>
81 /4 <i>id</i>	AND <i>r/m32</i> , <i>imm32</i>	<i>r/m32</i> AND <i>imm32</i>
83 /4 <i>ib</i>	AND <i>r/m16</i> , <i>imm8</i>	<i>r/m16</i> AND <i>imm8</i> (sign-extended)
83 /4 <i>ib</i>	AND <i>r/m32</i> , <i>imm8</i>	<i>r/m32</i> AND <i>imm8</i> (sign-extended)
20 /r	AND <i>r/m8</i> , <i>r8</i>	<i>r/m8</i> AND <i>r8</i>
21 /r	AND <i>r/m16</i> , <i>r16</i>	<i>r/m16</i> AND <i>r16</i>
21 /r	AND <i>r/m32</i> , <i>r32</i>	<i>r/m32</i> AND <i>r32</i>
22 /r	AND <i>r8</i> , <i>r/m8</i>	<i>r8</i> AND <i>r/m8</i>
23 /r	AND <i>r16</i> , <i>r/m16</i>	<i>r16</i> AND <i>r/m16</i>
23 /r	AND <i>r32</i> , <i>r/m32</i>	<i>r32</i> AND <i>r/m32</i>

Description

Performs a bitwise AND operation on the destination (first) and source (second) operands and stores the result in the destination operand location. The source operand can be an immediate, a register, or a memory location; the destination operand can be a register or a memory location. (However, two memory operands cannot be used in one instruction.) Each bit of the result of the AND instruction is a 1 if both corresponding bits of the operands are 1; otherwise, it becomes a 0.

Operation

$DEST \leftarrow DEST \text{ AND } SRC;$

Flags Affected

The OF and CF flags are cleared; the SF, ZF, and PF flags are set according to the result. The state of the AF flag is undefined.

Protected Mode Exceptions

#GP(0)	<p>If the destination operand points to a nonwritable segment.</p> <p>If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.</p> <p>If the DS, ES, FS, or GS register contains a null segment selector.</p>
#SS(0)	If a memory operand effective address is outside the SS segment limit.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made while the current privilege level is 3.

Real-Address Mode Exceptions

- #GP If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.
- #SS If a memory operand effective address is outside the SS segment limit.

Virtual-8086 Mode Exceptions

- #GP(0) If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.
- #SS(0) If a memory operand effective address is outside the SS segment limit.
- #PF(fault-code) If a page fault occurs.
- #AC(0) If alignment checking is enabled and an unaligned memory reference is made.

39.8 ARPL—Adjust RPL Field of Segment Selector

Opcode	Instruction	Description
63 /r	ARPL r/m16,r16	Adjust RPL of r/m16 to not less than RPL of r16

Description

Compares the RPL fields of two segment selectors. The first operand (the destination operand) contains one segment selector and the second operand (source operand) contains the other. (The RPL field is located in bits 0 and 1 of each operand.) If the RPL field of the destination operand is less than the RPL field of the source operand, the ZF flag is set and the RPL field of the destination operand is increased to match that of the source operand. Otherwise, the ZF flag is cleared and no change is made to the destination operand. (The destination operand can be a word register or a memory location; the source operand must be a word register.)

The ARPL instruction is provided for use by operating-system procedures (however, it can also be used by applications). It is generally used to adjust the RPL of a segment selector that has been passed to the operating system by an application program to match the privilege level of the application program. Here the segment selector passed to the operating system is placed in the destination operand and segment selector for the application program's code segment is placed in the source operand. (The RPL field in the source operand represents the privilege level of the application program.) Execution of the ARPL instruction then insures that the RPL of the segment selector received by the operating system is no lower (does not have a higher privilege) than the privilege level of the application program. (The segment selector for the application program's code segment can be read from the stack following a procedure call.)

See “Checking Caller Access Privileges” in Chapter 4 of the *Intel Architecture Software Developer's Manual, Volume 3*, for more information about the use of this instruction.

Operation

```
IF DEST(RPL) < SRC(RPL)
THEN
    ZF ← 1;
    DEST(RPL) ← SRC(RPL);
ELSE
    ZF ← 0;
FI;
```

Flags Affected

The ZF flag is set to 1 if the RPL field of the destination operand is less than that of the source operand; otherwise, is cleared to 0.

Protected Mode Exceptions

- | | |
|-----------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| #GP(0) | If the destination is located in a nonwritable segment.

If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

If the DS, ES, FS, or GS register is used to access memory and it contains a null segment selector. |
| #SS(0) | If a memory operand effective address is outside the SS segment limit. |
| #PF(fault-code) | If a page fault occurs. |
| #AC(0) | If alignment checking is enabled and an unaligned memory reference is made while the current privilege level is 3. |

Real-Address Mode Exceptions

- | | |
|-----|--------------------------------------------------------------|
| #UD | The ARPL instruction is not recognized in real-address mode. |
|-----|--------------------------------------------------------------|

Virtual-8086 Mode Exceptions

- | | |
|-----|--------------------------------------------------------------|
| #UD | The ARPL instruction is not recognized in virtual-8086 mode. |
|-----|--------------------------------------------------------------|

